IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BOARD OF PATENT APPEALS AND INTERFERENCES

In Re Application of:	)
	)
David H. Lin	) Confirmation No. 5529
	)
Serial No.:10/823,845	) Examiner: Ahluwalia, Navneet K.
	) Group Art Unit: 2166
Filed: April 14, 2004	)
	)
For: **Method and Apparatus for Multi-**	) HP Docket: 200402290-1
Process Access to a Linked List	) TKHR Docket: 050849-1390
	)

SUBSTITUTE APPEAL BRIEF UNDER 37 C.F.R. § 41.37

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

     This substitute Appeal Brief is submitted in response to the Notification of Non-

Compliant Appeal Brief mailed December 30, 2009, and in support of an appeal from the non-

final Office Action mailed May 14, 2009, rejecting claims 1-22 of the present application.

     The Notification indicated that the Argument section of the Appeal Brief filed July 20,

2009 was defective. Specifically, the Notification indicated that the rejection of claims 1-22 was

properly under § 103, whereas the Appeal Brief referred to a rejection under § 102. Appellant

submits this Substitute Appeal Brief remedies this deficiency.

TABLE OF CONTENTS

I.   **REAL PARTY IN INTEREST**

The real party in interest is Hewlett-Packard Development Company, LP, a limited

partnership established under the laws of the State of Texas and having a principal place of

business at 20555 S.H. 249 Houston, TX 77070, U.S.A. (hereinafter "HPDC"). HPDC is a Texas

limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware

Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ

Holdings, LLC.

II.   **RELATED APPEALS AND INTERFERENCES**

There are no related appeals or interferences.

III. **STATUS OF THE CLAIMS**

Claims 1-22 are pending in this application. Claims 1-22 stand twice rejected and are the

subject of this appeal.

IV. **STATUS OF AMENDMENTS**

There have been no claim amendments made after the final Office Action, and all

amendments made before the final Office Action have been entered. The claim listing in section

VIII. CLAIMS – APPENDIX (below) represents the present state of the claims.

V.   **SUMMARY OF THE CLAIMED SUBJECT MATTER**

Embodiments of the claimed subject matter are summarized below with reference

numbers and references to the written description ("specification") and drawings. The subject

matter described below appears in the original disclosure at least where indicated, and may

further appear in other places within the original disclosure.

Embodiments according to independent claim 1 involve a method for retrieving data. The

method comprises: locking a linked list (p. 6 lines 1-12; FIG. 1 element 5); retrieving data from

an element in the linked list and also advancing to a subsequent element while a breakpoint is

not encountered (p. 6 lines 1-12; p. 8 lines 3-13; FIG. 1 elements 5, 10, 15, 20, and 25); marking

the subsequent element in the linked-list as in-use when a breakpoint is encountered (p. 6 line

12 to p. 7 line 2; FIG. 1 element 15 and FIG. 2 element 30); creating a recommencement

reference to the subsequent element (p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40); and

unlocking the linked list (p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40).

     Embodiments according to independent claim 5 involve a method for deleting an

element from a linked list. The method comprises: determining if the element to be deleted is in-

use (p. 9 lines 1-22; FIG. 6 element 140); updating a recommencement reference to the

element to refer to a data element that is subsequent to the data element to be deleted when

the element in is in-use (p. 9 lines 5-12, lines 13-22; p. 9 line 28 to p. 10 line 19) and deleting

the element (p. 9 lines 1-12; FIG. 6 elements 145 and 155).

     Embodiments according to independent claim 7 involve an apparatus for storing and

retrieving data. The apparatus comprises: processor (p. 10 lines 20-22; FIG. 9 element 300)

capable of executing an instruction sequence (p. 11 lines 5-10); memory for storing an

instruction sequence (p. 10 lines 20-22; p. 11 lines 5-10; FIG. 9 element 370); input unit for

receiving data (p. 10 lines 20-25; FIG. 9 element 310); first output unit for providing data

according to a received data request (p. 10 lines 20-25; FIG. 9 element 320); one or more

ancillary output units for providing data according to a received data request; instruction

sequences stored in the memory (p. 10 lines 20-25) including: data storage module (p. 11 lines

15-16; FIG. 9 element 375) that, when executed by the processor, minimally causes the

processor to (p. 11 lines 5-15): receive data from the input unit (p. 11 lines 18-21); allocate a

data element to accommodate the data (p. 11 lines 21-23); create a reference to the data

element (p. 11 lines 23-25); store the reference in at least one of a header pointer and a forward

pointer included in a preceding data element (p. 11 line 23 to p. 12 line 2); and store the data in

the data element (p. 11 lines 25-28); data service module (p. 11 lines 15-16; FIG. 9 element

380) that, when executed by the processor, minimally causes the processor to (p. 11 lines

5-15): recognize a data request from the first output unit to the exclusion of all other data requests (p. 12 lines 3-6); provide data to the first output unit from a data element according to a data element reference (p. 12 lines 5-8) and also advance the data element reference to a subsequent data element while a breakpoint is not encountered (p. 12 lines 13-26); mark a subsequent data element as in-use when a breakpoint is encountered (p. 12 line 27 to p. 13 line 3; p. 13 lines 15-18); create a recommencement reference to a subsequent data element (p. 13 lines 3-7; p. 13 line 19 to p. 14 line 3); and enable recognition of other data requests (p. 13 lines 3-7).

Embodiments according to independent claim 13 involve a computer readable medium (p. 14 lines 18-28) having imparted thereon one or more instruction sequences for storing and retrieving data. The instruction sequences comprise: data storage module (p. 11 lines 15-16; FIG. 9 element 375) that, when executed by the processor, minimally causes the processor to (p. 11 lines 5-15): receive data from the input unit (p. 11 lines 18-21); allocate a data element to accommodate the data (p. 11 lines 21-23); create a reference to the data element (p. 11 lines 23-25); store the reference in at least one of a header pointer and a forward pointer included in a preceding data element (p. 11 line 23 to p. 12 line 2); and store the data in the data element (p. 11 lines 25-28); data service module (p. 11 lines 15-16; FIG. 9 element 380) that, when executed by the processor, minimally causes the processor to (p. 11 lines 5-15): recognize a data request from the first output unit to the exclusion of all other data requests (p. 12 lines 3-6); provide data to the first output unit from a data element according to a data element reference (p. 12 lines 5-8) and also advance the data element reference to a subsequent data element while a breakpoint is not encountered (p. 12 lines 13-26); mark a subsequent data element as in-use when a breakpoint is encountered (p. 12 line 27 to p. 13 line 3; p. 13 lines 15-18); create a recommencement reference to a subsequent data element (p. 13 lines 3-7; p. 13 line 19 to p. 14 line 3); and enable recognition of other data requests (p. 13 lines 3-7).

Embodiments according to independent claim 19 involve an apparatus for storing and retrieving data. The apparatus comprises: means for locking a linked list (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 lines 1-12; FIG. 1 element 5) means for retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 lines 1-12; p. 8 lines 3-13; FIG. 1 elements 5, 10, 15, 20, and 25); means for marking the subsequent element in the linked-list as in-use when a breakpoint is encountered (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 line 12 to p. 7 line 2; FIG. 1 element 15 and FIG. 2 element 30); means for creating a recommencement reference to the subsequent element (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40); and means for unlocking the linked list (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The following grounds of rejection are to be reviewed on appeal.

A.   Claims 1-22 stand rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over *Gao et al.* (U.S. 6,898,650) in view of *Marcotte* (U.S. 6,449,614).

VII. ARGUMENT

A. Rejection of Claims 1-22 under 35 U.S.C. §103(a): *Gao et al.* in view of *Marcotte*

Appellant submits that a *prima facie* case of obviousness for claims 1-22 has not been established using the references of record, for at least the following reasons. Therefore, Appellant requests that the rejection be overturned.

The U.S. Patent and Trademark Office bears the burden under 35 U.S.C. §103 to establish obviousness. *In re Fine*, 837 F.2d 1071, 1074, 5 U.S.P.Q. 2d 1596, 1598 (Fed. Cir. 1988). A proper rejection of a claim under 35 U.S.C. §103 as being obvious based upon a combination of references requires that the cited combination of references must disclose, teach, or suggest (either implicitly or explicitly) all elements/features/steps of the claim at issue.

See, e.g., In re Dow Chemical, 5 U.S.P.Q.2d 1529, 1531 (Fed. Cir. 1988); *In re Keller*, 208

U.S.P.Q.2d 871, 881 (C.C.P.A. 1981). Furthermore, even if the combination discloses all the

elements, "rejections on obviousness cannot be sustained by mere conclusory statements;

instead, there must be some articulated reasoning with some rational underpinning to support

the legal conclusion of obviousness." *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 418, 82

USPQ2d 1385, 1396 (2007)(quoting *In re Kahn*, 441 F.3d 977, 988, 78 USPQ2d 1329, 1336

(Fed. Cir. 2006)).

    1. **Independent Claim 1**

        a. **The combination does not teach
           "marking the subsequent element in the linked-list as in-use when a
           breakpoint is encountered"**

        The Office Action appears to rely (pp. 3–4) on *Gao et al.* for teaching "marking the

subsequent element in the linked-list as in-use", and Appellant assumes, for the sake of

argument, that *Gao et al.* teaching "marking the subsequent element in the linked-list as in-use".

The Office Action then acknowledges (p. 4) that *Gao et al.* does not disclose "the relinquishing

of control as a requirement as explicitly being claimed in the definition of breakpoint" and relies

(p. 4) on *Marcotte* to cure the admitted deficiency in *Gao et al.*, by allegedly teaching "the

condition where relinquishing of control may be required". Appellant disagrees with this

allegation.

        As an initial matter, claim 1 recites the condition "when a breakpoint is encountered",

and neither *Gao et al.* nor *Marcotte* uses the work "breakpoint" at all. Thus neither reference

explicitly teaches the claimed condition. The Examiner refers to "relinquishing of control as a

requirement as explicitly being claimed in the definition of breakpoint". Appellant notes that

"relinquishing of control as a requirement" is <u>not</u> explicitly claimed. The instant application does

state that "a breakpoint definition is used to define when a first process...is required to

relinquish control over the linked-list so that a second process can gain access to the linked list."

(Paragraph 10, emphasis added.) In addition, the instant application gives examples of how a breakpoint can be defined: establishing a maximum number of data elements that can be traversed in a single access session; and establishing a maximum time limit for a single access session. Appellant now assumes (for the sake of argument) that the claim construction apparently used by the Examiner is correct: that the combination must teach a process that marks the subsequent element in the linked-list as in-use <u>when</u> that process is <u>required</u> to relinquish control to another process.

As acknowledged by the Office Action (p. 4), *Gao et al.* does not teach "relinquishing of control as a requirement [of the marking action]". *Marcotte* is directed to the use of an exclusive lock to protect a data structure. The Office Action alleges (p. 4) that FIG. 2 and Col. 7 lines 26-65 of *Marcotte* discloses "how the lock that is hel[d] can be in a priority state and thus the queue would have to wait, similarly if the queue that was requiring the lock was in priority state the current item would have to relinquish the lock". Appellant disagrees with this characterization of the reference. The only discussion of priority in *Marcotte* is the following: "Referring to FIG. 2, state 101 includes a 32 bit word including E 121, W 122, L 123, P 124 and R 125 values, where…a bit in field L 123 indicates that the lock is held in a higher priority state…" Appellant submits that this brief mention of priority says nothing about waiting, and certainly does not teach that a task would be <u>required to wait</u> when the bit in field L 123 indicated that the lock is in a high priority state. Appellant submits that the Examiner's reading of this passage in *Marcotte* is beyond both the literal and implicit teachings of the reference. Rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand a lock held in a higher priority state to be the same as being the condition of being required to relinquish the lock, the Examiner has instead merely made a conclusory statement that the two are the same.

The relied-upon passage in *Marcotte* also teaches one task waiting on a lock that is held

by another task (*Marcotte*, Col. 7 lines 35-55). Appellant assumes that a person of ordinary skill

in the art would also understand this passage to teach, implicitly, that the lock holder eventually

relinquishes control to the waiter. Even so, this is not the same condition in which one task is

<u>required</u> to relinquish control to another task. Furthermore, even assuming that the Examiner's

characterization of *Marcotte* is accurate, so that *Marcotte* does teach that some tasks are

required to relinquish under some conditions, *Marcotte* does not teach or suggest the specific

feature of detecting and acting on this required-to-relinquish condition, as included in claim 1.

b. The combination does not teach "creating a recommencement reference to the subsequent element"

In alleging that *Gao et al.* discloses this feature, the Office Action refers to two different

sections of *Gao et al.*. However, neither of these sections discloses the claimed feature.

The Office Action first refers (p. 3) to Col. 4 lines 36-49 and 62-69 of *Gao et al.* Included

in this portion of *Gao et al.* is the text corresponding to block 510 of Figure 5. The only action

described in this portion of *Gao et al.* is setting the in-use flag under certain conditions (if it is not

set already). Appellant submits that setting the in-use flag is not the same as "creating a

recommencement reference to the subsequent element". Also included in the relied-upon

portion is the text corresponding to blocks 525, 530 and 535, which describes checking the valid

flag, unlocking the container, and using the container. The Examiner's allegation that any of

these teachings is the same as "creating a recommencement reference" is merely conclusory,

being unsupported by any evidence or reasoning as to why a person of ordinary skill in the art

would understand them to be the same.

The Office Action then further explains the rejection with a reference to a different

portion of *Gao et al.*, as follows:

> ...creating a recommencement reference to a subsquent element is
> clearly found in column 2 lines 46–58, column 3 lines 9-16 and lines 56-
> 59, when the breakpoint has been marked and the flag set as in-use ***the***

>*recommencement point is the one at breakpoint when the flag is
>unset or search for another container is made according to the
>algorithm in figure 5A & B*
>(Office Action, pp. 3-4, emphasis added.)

The Examiner thus appears to allege that the search for an in-use container stops at a particular

container, and that the search recommences at this point next time. Appellant disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be

readily verified by inspection of these two Figures, the entire sequence describes a single use of

the container, starting with acquiring the queue read/write lock (505) and ending with releasing

the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple

times by another piece of code, *the algorithm does not save state.* Without saved state, the

next invocation cannot recommence at a different point in the linked list. Therefore, the

algorithm does not create a reference to a recommencement point. The portions of *Gao et al.*

relied on here teach, at most, that the so-called "search" to locate a not-in-use container use

stops when it has found one. It does not teach that this state is saved, and therefore the search

cannot recommence again where it stopped before.

Appellant notes that an earlier Office Action (mailed November 6, 2008) also relied on

Gao et al. for teaching this feature, and it is not clear that the Examiner has withdrawn that

position. Therefore, Appellant now discusses those allegations about the claimed

"recommencement" feature. In that earlier Office Action (p. 4), the Examiner alleged "the

recommencement is clearly explained and showed by the pointer in the reference". The portions

of *Gao et al.* relied on here discuss two different pointers: a next pointer 220 in the queue head;

and a next pointer 325 in the container. Appellant must therefore assume that the Examiner is

specifically alleging that either next pointer 220 or next pointer 325 is a recommencement point.

However, *Gao et al.* merely teaches that next pointer 220 points to any one of the containers in

the queue (see Col. 2 lines 45-50; Col. 3 lines 9-16) and that next pointer 325 in each container

points to the next container (see Col. 3 lines 55-60). Neither of those uses of a next pointer has

anything to do with "a recommencement reference", must less the specific action of "creating a

recommencement reference to the subsequent element" as recited in claim 1. Once again, the

Examiner has merely made a conclusory statement that one of these next pointers is the same

as a "recommencement reference", rather than providing any reasoning or evidentiary

foundation as to why a person of ordinary skill in the art would understand these two to be the

same.

Marcotte does not disclose, teach, or suggest the "recommencement reference" feature

recited in claim 1, nor does the Office Action allege this is the case. Since the references do not

(individually or in combination) disclose, teach, or suggest this feature of claim 1, the rejection

should be overturned.

c. No motivation for combining *Gao et al.*'s in-use flag with *Marcotte*'s priority state

Appellant submits that it would <u>not</u> be obvious for one skilled in the art to combine *Gao*

et al. and *Marcotte* in a manner which results in the invention of claim 1. First, Appellant submits

that the proposed modification to *Gao et al.* – combining *Gao et al.*'s in-use flag with *Marcotte*'s

priority state – would render *Gao et al.* unsatisfactory for its intended purpose. *Gao et al.*

teaches specific conditions for setting the in-use flag to indicate that a container is locked –

which are *not the same* as the conditions recited in claim 1. (Specifically, the in-use flag will be

set if not set already, see Col. 4, lines 35-55, discussing atomic set-and-swap.) The Examiner

has not explained how *Gao et al.* would be modified to use *Marcotte*'s priority state as a

condition for in-use marking, rather than using set-if-not-set-already, and Appellant submits that

a simple substitution would render *Gao et al.* inoperable. If proposed modification would render

the prior art invention being modified unsatisfactory for its intended purpose, then there is no

suggestion or motivation to make the proposed modification. *In re Gordon*, 733 F.2d 900, 221

USPQ 1125 (Fed. Cir. 1984)

Second, Appellant submits that the motivation offered in the Office Action for combining the features of *Gao et al.* and *Marcotte* is deficient. The Office Action alleges that the "state of priority and the lock and its release capabilities of *Marcotte* reduce lock contention of *Gao et al.*'s method (column 4 lines 4-39, *Marcotte*). To begin, Appellant disagrees with this characterization of *Marcotte*. The cited portion of *Marcotte* actually teaches that a different feature – enabling a function to be executed after an update is made to the shared resource and the lock has been released – reduces lock contention. *Marcotte* says nothing about the advantages of the "state of priority and the lock and its release capabilities".

Finally, *Gao et al.* is directed to a "queuing method supporting multiple client accesses simultaneously", and so already addresses the problem of multiple clients contending for access to a shared resource. Appellant respectfully submits that it would <u>not</u> be obvious for one skilled in the art to look to a second reference to solve a problem already solved by a first reference. *See Ex parte Rinkevich*, 2007 WL 1552288 (BPAI May 29, 2007) (No. 2007-1317, Tech. Ctr. 2100). Accordingly, Appellant submits that the final Office Action fails to present a convincing line of reasoning as to why the artisan would have found the claimed invention to have been obvious as a combination of *Marcotte* and *Gao et al.* Rather, it appears that the only suggestion or motivation comes from Appellants' own disclosure. As is well established in the law, such hindsight to the Appellants' own disclosure is *per se* improper. *See Crown Operations International, Ltd. v. Solutia, Inc.*, 289 F.3d 1367, 62 USPQ2d 1917 (Fed. Cir. 2002) (a determination of obviousness cannot be based on a hindsight combination of components selectively culled from the prior art to fit the parameters of the invention).

d. Conclusion

As argued above, the combination does not disclose, teach, or suggest all elements of claim 1, and it would <u>not</u> be obvious for one skilled in the art to combine *Gao et al.* and *Marcotte*

in a manner which results in the invention of claim 1. For at least these reasons, a *prima facie*

case of obviousness has not been made, and the rejection of claim 1 should be withdrawn.

2. Independent Claim 5

a. *Gao et al.* and *Marcotte* does not teach "updating a recommencement reference to the element to refer to a data element that is subsequent to the data element to be deleted when the element in is in-use"

The Office Action specifically alleges (p. 5) that *Gao et al.* discloses this feature in Table

14. Appellant disagrees. This portion of *Gao et al.* is "pseudo-code of the process for removing

data from a container not currently in use" (Col. 5, lines 15-20). Appellant submits that removing

data from a container not currently in use is not the same as "updating a recommencement

reference to the element to refer to a data element that is subsequent to the data element to be

deleted when the element in is in-use". The Examiner has merely made a conclusory statement

that the two actions are the same, rather than providing any reasoning or evidentiary foundation

as to why a person of ordinary skill in the art would understand these two to be the same.

Appellant notes that claim 1 also includes the phrase "recommencement reference", and

in rejecting this feature of claim 1 the Office Action (pp. 3-4) relies on a different portion of *Gao

et al.*, Col. 2 lines 46 – 58, Col. 3 lines 9-16, and Col. 3 lines 56-59. However, Appellant submits

that this portion of *Gao et al.* also fails to teach a "recommencement reference" or "updating a

recommencement reference". That section of the Office Action states that:

> …creating a recommencement reference to a subsquent element in Gao
> is clearly found in column 2 lines 46 – 58, column 3 lines 9-16 and lines
> 56-59, when the breakpoint has been marked and the flag set as in-use
> **the recommencement point is the one at breakpoint when the flag is
> unset or search for another container is made according to the
> algorithm in figure 5A & B.**
> (Office Action, pp. 3-4, emphasis added.)

Thus, the Examiner appears to allege that the search for an in-use container stops at a

particular container, and that the search recommences at this point next time. Appellant

disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be readily verified by inspection of these two Figures, the entire sequence describes a single use of the container, starting with acquiring the queue read/write lock (505) and ending with releasing the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple times by another piece of code, *the algorithm does not save state.* Without saved state, the next invocation cannot recommence at a different point in the linked list. Therefore, the algorithm does not create a reference to a recommencement point. The portions of *Gao et al.* relied on here teach, at most, that the so-called "search" to locate a not-in-use container use stops when it has found one. It does not teach that this state is saved, and therefore the search cannot recommence again where it stopped before.

Marcotte does not disclose, teach, or suggest the "recommencement reference" feature recited in claim 5, nor does the Office Action allege this is the case. Since the references do not (individually or in combination) disclose, teach, or suggest this feature of claim 5, the rejection should be overturned.

b. No motivation for combining *Gao et al.*'s in-use flag with *Marcotte*'s priority state

Appellant submits that it would <u>not</u> be obvious for one skilled in the art to combine *Gao et al.* and *Marcotte* in a manner which results in the invention of claim 5. To begin, Appellant submits that the motivation offered in the Office Action for combining the features of *Gao et al.* and *Marcotte* is deficient. The Office Action alleges that the "state of priority and the lock and its release capabilities of *Marcotte* reduce lock contention of *Gao et al.*'s method (column 4 lines 4-39, *Marcotte*). To begin, Appellant disagrees with this characterization of *Marcotte*. The cited portion of *Marcotte* actually teaches that a different feature – enabling a function to be executed after an update is made to the shared resource and the lock has been released – reduces lock

contention. *Marcotte* says nothing about the advantages of the "state of priority and the lock and its release capabilities".

Furthermore, *Gao et al.* is directed to a "queuing method supporting multiple client accesses simultaneously", and so already addresses the problem of multiple clients contending for access to a shared resource. Appellant respectfully submits that it would <u>not</u> be obvious for one skilled in the art to look to a second reference to solve a problem already solved by a first reference. *See Ex parte Rinkevich*, 2007 WL 1552288 (BPAI May 29, 2007) (No. 2007-1317, Tech. Ctr. 2100). Accordingly, Appellant submits that the final Office Action fails to present a convincing line of reasoning as to why the artisan would have found the claimed invention to have been obvious as a combination of *Marcotte* and *Gao et al.* Rather, it appears that the only suggestion or motivation comes from Appellants' own disclosure. As is well established in the law, such hindsight to the Appellants' own disclosure is *per se* improper. *See Crown Operations International, Ltd. v. Solutia, Inc.*, 289 F.3d 1367, 62 USPQ2d 1917 (Fed. Cir. 2002) (a determination of obviousness cannot be based on a hindsight combination of components selectively culled from the prior art to fit the parameters of the invention).

c. Conclusion

As argued above, the combination does not disclose, teach, or suggest all elements of claim 5, and it would <u>not</u> be obvious for one skilled in the art to combine *Gao et al.* and *Marcotte* in a manner which results in the invention of claim 5. For at least these reasons, a *prima facie* case of obviousness has not been made, and the rejection of claim 5 should be withdrawn.

3. Independent Claims 7 and 13

a. The combination does not teach
"data storage module that, when executed by the processor, minimally causes the processor to…mark a subsequent data element as in-use when a breakpoint is encountered"

The Office Action appears to rely (pp. 7-8 and 10-12) on *Gao et al.* for teaching "marking the subsequent element in the linked-list as in-use", and Appellant assumes, for the sake of

argument, that *Gao et al.* teaches "marking the subsequent element in the linked-list as in-use". The Office Action then acknowledges (p. 8 and pp. 11-12) that *Gao et al.* does not disclose "the relinquishing of control as a requirement as explicitly being claimed in the definition of breakpoint" and relies (p. 8 and p. 11) on *Marcotte* to cure the admitted deficiency in *Gao et al.*, by allegedly teaching "the condition where relinquishing of control may be required". Appellant disagrees with this allegation.

As an initial matter, claims 7 and 13 recite the condition "when a breakpoint is encountered", and neither *Gao et al.* nor *Marcotte* uses the work "breakpoint" at all. Thus neither reference explicitly teaches the claimed condition. The Examiner refers to "relinquishing of control as a requirement as explicitly being claimed in the definition of breakpoint". Appellant notes that "relinquishing of control as a requirement" is <u>not</u> explicitly claimed. The instant application does state that "a breakpoint definition is used to define when a first process…is required to relinquish control over the linked-list so that a second process can gain access to the linked list." (Paragraph 10, emphasis added.) In addition, the instant application gives examples of how a breakpoint can be defined: establishing a maximum number of data elements that can be traversed in a single access session; and establishing a maximum time limit for a single access session. Appellant now assumes (for the sake of argument) that the claim construction apparently used by the Examiner is correct: that the combination must teach a process that marks the subsequent element in the linked-list as in-use <u>when</u> that process is <u>required</u> to relinquish control to another process.

As acknowledged by the Office Action (p. 8 and p. 12), *Gao et al.* does not teach "relinquishing of control as a requirement [of the marking action]". *Marcotte* is directed to the use of an exclusive lock to protect a data structure. The Office Action alleges (p. 8 and pp. 11-12) that FIG. 2 and Col. 7 lines 26-65 of *Marcotte* discloses "how the lock that is hel[d] can be in a priority state and thus the queue would have to wait, similarly if the queue that was requiring the

lock was in priority state the current item would have to relinquish the lock". Appellant disagrees with this characterization of the reference. The only discussion of priority in *Marcotte* is the following: "Referring to FIG. 2, state 101 includes a 32 bit word including E 121, W 122, L 123, P 124 and R 125 values, where…a bit in field L 123 indicates that the lock is held in a higher priority state…" Appellant submits that this brief mention of priority says nothing about waiting, and certainly does not teach that a task would be <u>required to wait</u> when the bit in field L 123 indicated that the lock is in a high priority state. Appellant submits that the Examiner's reading of this passage in *Marcotte* is beyond both the literal and implicit teachings of the reference. Rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand a lock held in a higher priority state to be the same as being the condition of being required to relinquish the lock, the Examiner has instead merely made a conclusory statement that the two are the same.

The relied-upon passage in *Marcotte* also teaches one task waiting on a lock that is held by another task (*Marcotte*, Col. 7 lines 35-55). Appellant assumes that a person of ordinary skill in the art would also understand this passage to teach, implicitly, that the lock holder eventually relinquishes control to the waiter. Even so, this is not the same the condition in which one task is <u>required</u> to relinquish control to another task. Furthermore, even assuming that the Examiner's characterization of *Marcotte* is accurate, so that *Marcotte* does teach that some tasks are required to relinquish under some conditions, *Marcotte* does not teach or suggest the specific feature of detecting and acting on this required-to-relinquish condition, as included in claims 7 and 13.

**b. The combination does not teach
"creating a recommencement reference to the subsequent element"**

In alleging that *Gao et al.* discloses this feature, the Office Action refers to two different sections of *Gao et al.* However, neither of these sections discloses the claimed feature.

The Office Action first refers (p. 3) to Col. 4 lines 36-49 and 62-69 of *Gao et al.* Included in this portion of *Gao et al.* is the text corresponding to block 510 of Figure 5. The only action described in this portion of *Gao et al.* is setting the in-use flag under certain conditions (if it is not set already). Appellant submits that setting the in-use flag is not the same as "creating a recommencement reference to the subsequent element". Also included in the relied-upon portion is the text corresponding to blocks 525, 530 and 535, which describes checking the valid flag, unlocking the container, and using the container. The Examiner's allegation that any of these teachings is the same as "creating a recommencement reference" is merely conclusory, being unsupported by any evidence or reasoning as to why a person of ordinary skill in the art would understand them to be the same.

The Office Action then further explains the rejection with a reference to a different portion of *Gao et al.*, as follows:

> ...creating a recommencement reference to a subsquent element in Gao is clearly found in column 2 lines 46–58, column 3 lines 9-16 and lines 56-59, when the breakpoint has been marked and the flag set as in-use **the recommencement point is the one at breakpoint when the flag is unset or search for another container is made according to the algorithm in figure 5A & B.**
> (Office Action, pp. 7-8, emphasis added.)

The Examiner thus appears to allege that the search for an in-use container stops at a particular container, and that the search recommences at this point next time. Appellant disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be readily verified by inspection of these two Figures, the entire sequence describes a single use of the container, starting with acquiring the queue read/write lock (505) and ending with releasing the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple times by another piece of code, **the algorithm does not save state.** Without saved state, the next invocation cannot recommence at a different point in the linked list. Therefore, the

algorithm does not create a reference to a recommencement point. The portions of *Gao et al.* relied on here teach, at most, that the so-called "search" to locate a not-in-use container use stops when it has found one. It does not teach that this state is saved, and therefore the search cannot recommence again where it stopped before.

Appellant notes that an earlier Office Action (mailed November 6, 2008) also relied on *Gao et al.* for teaching this feature, and it is not clear that the Examiner has withdrawn that position. Therefore, Appellant now discusses those allegations about the claimed "recommencement" feature. In that earlier Office Action (p. 4), the Examiner alleged "the recommencement is clearly explained and showed by the pointer in the reference". The portions of *Gao et al.* relied on here discuss two different pointers: a next pointer 220 in the queue head; and a next pointer 325 in the container. Appellant must therefore assume that the Examiner is specifically alleging that either next pointer 220 or next pointer 325 is a recommencement point. However, *Gao et al.* merely teaches that next pointer 220 points to any one of the containers in the queue (see Col. 2 lines 45-50; Col. 3 lines 9-16) and that next pointer 325 in each container points to the next container (see Col. 3 lines 55-60). Neither of those uses of a next pointer has anything to do with "a recommencement reference", must less the specific action of "creating a recommencement reference to the subsequent element" as recited in claims 7 and 13. Once again, the Examiner has merely made a conclusory statement that one of these next pointers is the same as a "recommencement reference", rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

Marcotte does not disclose, teach, or suggest the "recommencement reference" feature recited in claims 7 and 13, nor does the Office Action allege this is the case. Since the references do not (individually or in combination) disclose, teach, or suggest this feature of claims 7 and 13, the rejection should be overturned.

c. **No motivation for combining *Gao et al.*'s in-use flag with *Marcotte*'s priority state**

Appellant submits that it would <u>not</u> be obvious for one skilled in the art to combine *Gao et al.* and *Marcotte* in a manner which results in the invention of claims 7 and 13. First, Appellant submits that the proposed modification to *Gao et al.* – combining *Gao et al.*'s in-use flag with *Marcotte*'s priority state – would render *Gao et al.* unsatisfactory for its intended purpose. *Gao et al.* teaches specific conditions for setting the in-use flag to indicate that a container is locked – which are ***not the same*** as the conditions recited in claims 7 and 13. (Specifically, the in-use flag will be set if not set already, see Col. 4, lines 35-55, discussing atomic set-and-swap.) The Examiner has not explained how *Gao et al.* would be modified to use *Marcotte*'s priority state as a condition for in-use marking, rather than using set-if-not-set-already, and Appellant submits that a simple substitution would render *Gao et al.* inoperable. If proposed modification would render the prior art invention being modified unsatisfactory for its intended purpose, then there is no suggestion or motivation to make the proposed modification. *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984).

Second, Appellant submits that the motivation offered in the Office Action for combining the features of *Gao et al.* and *Marcotte* is deficient. The Office Action alleges that the "state of priority and the lock and its release capabilities of *Marcotte* reduce lock contention of *Gao et al.*'s method (column 4 lines 4-39, *Marcotte*). To begin, Appellant disagrees with this characterization of *Marcotte*. The cited portion of *Marcotte* actually teaches that a different feature – enabling a function to be executed after an update is made to the shared resource and the lock has been released – reduces lock contention. *Marcotte* says nothing about the advantages of the "state of priority and the lock and its release capabilities".

Finally, *Gao et al.* is directed to a "queuing method supporting multiple client accesses simultaneously", and so already addresses the problem of multiple clients contending for access to a shared resource. Appellant respectfully submits that it would <u>not</u> be obvious for one skilled

in the art to look to a second reference to solve a problem already solved by a first reference.

See Ex parte Rinkevich, 2007 WL 1552288 (BPAI May 29, 2007) (No. 2007-1317, Tech. Ctr.

2100). Accordingly, Appellant submits that the final Office Action fails to present a convincing

line of reasoning as to why the artisan would have found the claimed invention to have been

obvious as a combination of *Marcotte* and *Gao et al.* Rather, it appears that the only suggestion

or motivation comes from Appellants' own disclosure. As is well established in the law, such

hindsight to the Appellants' own disclosure is *per se* improper. *See Crown Operations*

International, Ltd. v. Solutia, Inc., 289 F.3d 1367, 62 USPQ2d 1917 (Fed. Cir. 2002) (a

determination of obviousness cannot be based on a hindsight combination of components

selectively culled from the prior art to fit the parameters of the invention).

   d. Conclusion

   As argued above, the combination does not disclose, teach, or suggest all elements of

claims 7 and 13, and it would <u>not</u> be obvious for one skilled in the art to combine *Gao et al.* and

Marcotte in a manner which results in the invention of claims 7 and 13. For at least these

reasons, a *prima facie* case of obviousness has not been made, and the rejection of claims 7

and 13 should be withdrawn.

   4. Independent Claim 19

   a. The combination does not teach
   "means for marking the subsequent element in the linked-list as in-use when a
   breakpoint is encountered "

   The Office Action appears to rely (p. 14) on *Gao et al.* for teaching "marking the

subsequent element in the linked-list as in-use", and Appellant assumes, for the sake of

argument, that *Gao et al.* teaches "marking the subsequent element in the linked-list as in-use".

The Office Action then acknowledges (p. 15) that *Gao et al.* does not disclose "the relinquishing

of control as a requirement as explicitly being claimed in the definition of breakpoint" and relies

(p. 15) on *Marcotte* to cure the admitted deficiency in *Gao et al.*, by allegedly teaching "the

condition where relinquishing of control may be required". Appellant disagrees with this allegation.

As an initial matter, claim 19 recites the condition "when a breakpoint is encountered", and neither *Gao et al.* nor *Marcotte* uses the work "breakpoint" at all. Thus neither reference explicitly teaches the claimed condition. The Examiner refers to "relinquishing of control as a requirement as explicitly being claimed in the definition of breakpoint". Appellant notes that "relinquishing of control as a requirement" is <u>not</u> explicitly claimed. The instant application does state that "a breakpoint definition is used to define when a first process…is required to relinquish control over the linked-list so that a second process can gain access to the linked list." (Paragraph 10, emphasis added.) In addition, the instant application gives examples of how a breakpoint can be defined: establishing a maximum number of data elements that can be traversed in a single access session; and establishing a maximum time limit for a single access session. Appellant now assumes (for the sake of argument) that the claim construction apparently used by the Examiner is correct: that the combination must teach a process that marks the subsequent element in the linked-list as in-use <u>when</u> that process is <u>required</u> to relinquish control to another process.

As acknowledged by the Office Action (p. 14), *Gao et al.* does not teach "relinquishing of control as a requirement [of the marking action]". *Marcotte* is directed to the use of an exclusive lock to protect a data structure. The Office Action alleges (p. 4) that FIG. 2 and Col. 7 lines 26-65 of *Marcotte* discloses "how the lock that is hel[d] can be in a priority state and thus the queue would have to wait, similarly if the queue that was requiring the lock was in priority state the current item would have to relinquish the lock". Appellant disagrees with this characterization of the reference. The only discussion of priority in *Marcotte* is the following: "Referring to FIG. 2, state 101 includes a 32 bit word including E 121, W 122, L 123, P 124 and R 125 values, where…a bit in field L 123 indicates that the lock is held in a higher priority

state…" Appellant submits that this brief mention of priority says nothing about waiting, and certainly does not teach that a task would be <u>required to wait</u> when the bit in field L 123 indicated that the lock is in a high priority state. Appellant submits that the Examiner's reading of this passage in *Marcotte* is beyond both the literal and implicit teachings of the reference. Rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand a lock held in a higher priority state to be the same as being the condition of being required to relinquish the lock, the Examiner has instead merely made a conclusory statement that the two are the same.

The relied-upon passage in *Marcotte* also teaches one task waiting on a lock that is held by another task (*Marcotte*, Col. 7 lines 35-55). Appellant assumes that a person of ordinary skill in the art would also understand this passage to teach, implicitly, that the lock holder eventually relinquishes control to the waiter. Even so, this is not the same the condition in which one task is <u>required</u> to relinquish control to another task. Furthermore, even assuming that the Examiner's characterization of *Marcotte* is accurate, so that *Marcotte* does teach that some tasks are required to relinquish under some conditions, *Marcotte* does not teach or suggest the specific feature of detecting and acting on this required-to-relinquish condition, as included in claim 19.

**b. The combination does not teach
"means for creating a recommencement reference to the subsequent element"**

In alleging that *Gao et al.* discloses this feature, the Office Action refers to two different sections of *Gao et al.*. However, neither of these sections discloses the claimed feature.

The Office Action first refers (p. 3) to Col. 4 lines 36-49 and 62-69 of *Gao et al.* Included in this portion of *Gao et al.* is the text corresponding to block 510 of Figure 5. The only action described in this portion of *Gao et al.* is setting the in-use flag under certain conditions (if it is not set already). Appellant submits that setting the in-use flag is not the same as "creating a recommencement reference to the subsequent element". Also included in the relied-upon

portion is the text corresponding to blocks 525, 530 and 535, which describes checking the valid flag, unlocking the container, and using the container. The Examiner's allegation that any of these teachings is the same as "creating a recommencement reference" is merely conclusory, being unsupported by any evidence or reasoning as to why a person of ordinary skill in the art would understand them to be the same.

The Office Action then further explains the rejection with a reference to a different portion of *Gao et al.*, as follows:

> …creating a recommencement reference to a subsquent element in Gao is clearly found in column 2 lines 46–58, column 3 lines 9-16 and lines 56-59, when the breakpoint has been marked and the flag set as in-use **the recommencement point is the one at breakpoint when the flag is unset or search for another container is made according to the algorithm in figure 5A & B**
> (Office Action, pp. 3-4, emphasis added.)

The Examiner thus appears to allege that the search for an in-use container stops at a particular container, and that the search recommences at this point next time. Appellant disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be readily verified by inspection of these two Figures, the entire sequence describes a single use of the container, starting with acquiring the queue read/write lock (505) and ending with releasing the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple times by another piece of code, **the algorithm does not save state**. Without saved state, the next invocation cannot recommence at a different point in the linked list. Therefore, the algorithm does not create a reference to a recommencement point. The portions of *Gao et al.* relied on here teach, at most, that the so-called "search" to locate a not-in-use container use stops when it has found one. It does not teach that this state is saved, and therefore the search cannot recommence again where it stopped before.

Appellant notes that an earlier Office Action (mailed November 6, 2008) also relied on *Gao et al.* for teaching this feature, and it is not clear that the Examiner has withdrawn that position. Therefore, Appellant now discusses those allegations about the claimed "recommencement" feature. In that earlier Office Action (p. 4), the Examiner alleged "the recommencement is clearly explained and showed by the pointer in the reference". The portions of *Gao et al.* relied on in that earlier Office Action discuss two different pointers: a next pointer 220 in the queue head; and a next pointer 325 in the container. Appellant must therefore assume that the Examiner is specifically alleging that either next pointer 220 or next pointer 325 is a recommencement point. However, *Gao et al.* merely teaches that next pointer 220 points to any one of the containers in the queue (see Col. 2 lines 45-50; Col. 3 lines 9-16) and that next pointer 325 in each container points to the next container (see Col. 3 lines 55-60). Neither of those uses of a next pointer has anything to do with "a recommencement reference", must less the specific action of "creating a recommencement reference to the subsequent element" as recited in claim 19. Once again, the Examiner has merely made a conclusory statement that one of these next pointers is the same as a "recommencement reference", rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

Marcotte does not disclose, teach, or suggest the "recommencement reference" feature recited in claim 19, nor does the Office Action allege this is the case. Since the references do not (individually or in combination) disclose, teach, or suggest this feature of claim 19, the rejection should be overturned.

c. No motivation for combining *Gao et al.*'s in-use flag with *Marcotte*'s priority state

Appellant submits that it would <u>not</u> be obvious for one skilled in the art to combine *Gao et al.* and *Marcotte* in a manner which results in the invention of claim 19. First, Appellant submits that the proposed modification to *Gao et al.* – combining *Gao et al.*'s in-use flag with

Marcotte's priority state – would render *Gao et al.* unsatisfactory for its intended purpose. *Gao et al.* teaches specific conditions for setting the in-use flag to indicate that a container is locked – which are **not the same** as the conditions recited in claim 19. (Specifically, the in-use flag will be set if not set already, see Col. 4, lines 35-55, discussing atomic set-and-swap.) The Examiner has not explained how *Gao et al.* would be modified to use *Marcotte*'s priority state as a condition for in-use marking, rather than using set-if-not-set-already, and Appellant submits that a simple substitution would render *Gao et al.* inoperable. If proposed modification would render the prior art invention being modified unsatisfactory for its intended purpose, then there is no suggestion or motivation to make the proposed modification. *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984)

Second, Appellant submits that the motivation offered in the Office Action for combining the features of *Gao et al.* and *Marcotte* is deficient. The Office Action alleges that the "state of priority and the lock and its release capabilities of *Marcotte* reduce lock contention of *Gao et al.*'s method (column 4 lines 4-39, *Marcotte*). To begin, Appellant disagrees with this characterization of *Marcotte*. The cited portion of *Marcotte* actually teaches that a different feature – enabling a function to be executed after an update is made to the shared resource and the lock has been released – reduces lock contention. *Marcotte* says nothing about the advantages of the "state of priority and the lock and its release capabilities".

Finally, *Gao et al.* is directed to a "queuing method supporting multiple client accesses simultaneously", and so already addresses the problem of multiple clients contending for access to a shared resource. Appellant respectfully submits that it would <u>not</u> be obvious for one skilled in the art to look to a second reference to solve a problem already solved by a first reference. *See Ex parte Rinkevich*, 2007 WL 1552288 (BPAI May 29, 2007) (No. 2007-1317, Tech. Ctr. 2100). Accordingly, Appellant submits that the final Office Action fails to present a convincing line of reasoning as to why the artisan would have found the claimed invention to have been

obvious as a combination of *Marcotte* and *Gao et al.* Rather, it appears that the only suggestion or motivation comes from Appellants' own disclosure. As is well established in the law, such hindsight to the Appellants' own disclosure is *per se* improper. *See Crown Operations International, Ltd. v. Solutia, Inc.*, 289 F.3d 1367, 62 USPQ2d 1917 (Fed. Cir. 2002) (a determination of obviousness cannot be based on a hindsight combination of components selectively culled from the prior art to fit the parameters of the invention).

d. Conclusion

As argued above, the combination does not disclose, teach, or suggest all elements of claim 19, and it would <u>not</u> be obvious for one skilled in the art to combine *Gao et al.* and *Marcotte* in a manner which results in the invention of claim 19. For at least these reasons, a *prima facie* case of obviousness has not been made, and the rejection of claim 19 should be withdrawn.

5. Dependent Claims 2-6, 8-12, 14-18, and 20-22

Since independent claims 1, 5, 7, 13, and 19 are allowable, Appellant submits that claims 2-6, 8-12, 14-18, and 20-22 are allowable for at least the reason that each depends from an allowable claim. *In re Fine*, 837 F.2d 1071, 5 U.S.P.Q. 2d 1596, 1598 (Fed. Cir. 1988). Therefore, Appellant requests that the rejection of claims 2-6, 8-12, 14-18, and 20-22 be overturned. .

CONCLUSION

For at least the reasons discussed above, Appellant respectfully requests that the Examiner's final rejection of claims 1-22 be overturned by the Board. In addition to the claims listed in Section VIII (CLAIMS – APPENDIX), Section IX (EVIDENCE – APPENDIX) included herein indicates that there is no additional evidence relied upon by this brief. Section X (RELATED PROCEEDINGS – APPENDIX) included herein indicates that there are no related proceedings.

Respectfully submitted,

By: /Karen G. Hazzah/

Karen G. Hazzah,
Reg. No. 48,472

THOMAS, KAYDEN, HORSTEMEYER
& RISLEY, L.L.P.

600 Galleria Parkway, SE
Suite 1500
Atlanta, Georgia 30339-5948
Tel: (770) 933-9500
Fax: (770) 951-0933

VIII.   CLAIMS – APPENDIX

1. A method for retrieving data comprising:

locking a linked list;

retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered;

marking the subsequent element in the linked-list as in-use when a breakpoint is encountered;

creating a recommencement reference to the subsequent element; and

unlocking the linked list.

2. The method of claim 1 further comprising:

locking the linked list;

determining a subsequent element in the linked list according to the recommencement reference; and

retrieving data from the determined subsequent element.

3. The method of claim 1 wherein creating a recommencement reference to the subsequent element comprises:

retrieving a pointer to the subsequent element;

determining a process identifier for a current process; and

associating the pointer with the process identifier.

4. The method of claim 1 wherein marking the subsequent element in the linked-list as in-use comprises maintaining a count of the quantity of processes that require additional access to the element.

5. A method for deleting an element from a linked list comprising:

determining if the element to be deleted is in-use;

updating a recommencement reference to the element to refer to a data element that is subsequent to the data element to be deleted when the element in is in-use; and

deleting the element.

6. The method of claim 5 wherein updating a recommencement reference to the element comprises:

discovering a pointer associated with a process identifier;

disassociating the process identifier from the pointer;

determining a pointer to a subsequent element; and

associating the process identifier with the newly determined pointer.

7. An apparatus for storing and retrieving data comprising:

processor capable of executing an instruction sequence;

memory for storing an instruction sequence;

input unit for receiving data;

first output unit for providing data according to a received data request;

one or more ancillary output units for providing data according to a received data request;

instruction sequences stored in the memory including:

data storage module that, when executed by the processor, minimally causes the processor to:

receive data from the input unit;

allocate a data element to accommodate the data;

create a reference to the data element;

store the reference in at least one of a header pointer and a forward pointer included in a preceding data element; and

store the data in the data element;

data service module that, when executed by the processor, minimally causes the processor to:

recognize a data request from the first output unit to the exclusion of all other data requests;

provide data to the first output unit from a data element according to a data element reference and also advance the data element reference to a subsequent data element while a breakpoint is not encountered;

mark a subsequent data element as in-use when a breakpoint is encountered;

create a recommencement reference to a subsequent data element; and

enable recognition of other data requests.

8. The apparatus of claim 7 wherein the data service module, when executed by the processor, further minimally causes the processor to:

recognize a data request from the first output unit to the exclusion of all other data requests; and

provide data to the first output unit from a data element according to the recommencement reference.

9. The apparatus of claim 7 wherein the data service module causes the processor to create a recommencement reference by minimally causing the processor to:

retrieve a pointer to a data element subsequent to a current data element;

determine an identifier associated with the data request received from the first output unit; and

store the retrieved pointer and the determined identifier in an associative manner.

10. The apparatus of claim 7 wherein the data service module causes the processor to mark a subsequent data element as in-use by minimally causing the processor to increment a use counter included in a subsequent data element.

11. The apparatus of claim 7 wherein the data service module further minimally causes the processor to receive a delete data request from an output unit by minimally causing the processor to:

determine if a data element to be deleted is in-use;

update a recommencement reference to refer to a data element that is subsequent to the data element to be deleted; and

delete the data element according to the received delete data request.

12. The apparatus of claim 11 wherein the data service module causes the processor to update a recommencement reference by minimally causing the processor to:

discover a pointer according to a data request identifier; and

replace the pointer with a pointer to a data element that is subsequent to the data element to be deleted.

13. A computer readable medium having imparted thereon one or more instruction sequences for storing and retrieving data comprising:

data storage module that, when executed by a processor, minimally causes the processor to:

receive data from an input unit;

allocate a data element to accommodate the data;

create a reference to the data element;

store the reference in at least one of a header pointer and a forward pointer included in a preceding data element; and

store the data in the data element;

data service module that, when executed by a processor, minimally causes the processor to:

recognize a data request from a first output unit to the exclusion of all other data requests;

provide data to a first output unit from a data element according to a data element reference and also advance the data element reference to a subsequent data element while a breakpoint is not encountered;

mark a subsequent data element as in-use when a breakpoint is encountered;

create a recommencement reference to a subsequent data element; and

enable recognition of other data requests.

14. The computer readable medium of claim 13 wherein the data service module, when executed by a processor, further minimally causes the processor to:

recognize a data request from a first output unit to the exclusion of all other data requests; and

provide data to a first output unit from a data element according to the recommencement reference.

15. The computer readable medium of claim 13 wherein the data service module causes a processor to create a recommencement reference by minimally causing the processor to:

retrieve a pointer to a data element subsequent to a current data element;

determine an identifier associated with a data request received from a first output unit; and

store the retrieved pointer and the determined identifier in an associative manner.

16. The computer readable medium of claim 13 wherein the data service module causes a processor to mark a subsequent data element as in-use by minimally causing the processor to increment a use counter included in a subsequent data element.

17. The computer readable medium of claim 13 wherein the data service module further minimally causes the processor to receive a delete data request from an output unit by minimally causing the processor to:

determine if a data element to be deleted is in-use;

update a recommencement reference to refer to a data element that is subsequent to the data element to be deleted; and

delete the data element according to the received delete data request.

18. The computer readable medium of claim 17 wherein the data service module causes the processor to update a recommencement reference by minimally causing the processor to:

discover a pointer according to a data request identifier; and

replace the pointer with a pointer to a data element that is subsequent to the data element to be deleted.

19. An apparatus for storing and retrieving data comprising:

    means for locking a linked list;

    means for retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered;

    means for marking the subsequent element in the linked-list as in-use when a breakpoint is encountered;

    means for creating a recommencement reference to the subsequent element; and

    means for unlocking the linked list.

20. The apparatus of claim 19 further comprising:

    means for locking the linked list;

    means for determining a subsequent element in the linked list according to the recommencement reference; and

    means for retrieving data from the determined subsequent element.

21. The apparatus of claim 19 further comprising a means for deleting an element in the linked-list.

22. The apparatus of claim 21 wherein the means for deleting an element comprises:

    means for determining if the element to be deleted is in-use;

    means for updating a reference to the element to refer to a subsequent element in the linked list when the element in is in-use; and

    means for deleting the element.

IX. **EVIDENCE – APPENDIX**

None.

X. **RELATED PROCEEDINGS – APPENDIX**

None.